

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☒ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

PUB. NO.: 09-311786 [JP 9311786 A]

PUBLISHED: December 02, 1997 (19971202)

INVENTOR(s): UMINAGA MASAHIRO

SAITO YASUHIKO

APPLICANT(s): HITACHI LTD [000510] (A Japanese Company or Corporation), JP
(Japan)

APPL. NO.: 09-052772 [JP 9752772]

FILED: March 07, 1997 (19970307)

INTL CLASS: [6] G06F-009/38

JAPIO CLASS: 45.1 (INFORMATION PROCESSING -- Arithmetic Sequence Units)

JAPIO KEYWORD: R131 (INFORMATION PROCESSING -- Microcomputers &
Microprocessors)

ABSTRACT

PROBLEM TO BE SOLVED: To reduce a pipeline stall due to a data hazard of a superscalar system and to improve the processing speed by changing an instruction in 1st instruction format stored in an instruction memory into an instruction in 2nd instruction format.

SOLUTION: The instruction is taken in a 1st stage from the instruction memory and the instruction taken in the 1st stage 101 is decoded in a 2nd stage 103. The decoded instruction is executed in a 3rd stage and when the execution result is written in a register in a 4th stage 107, the instruction in the 1st instruction format stored in the instruction memory is changed into the instruction in the 2nd instruction format and executed. Consequently, the pipeline stall due to the data hazard of the superscalar system can be reduced and the processing speed is improved.

(51)Int.Cl.⁶

G 0 6 F 9/38

識別記号

3 1 0

庁内整理番号

F I

G 0 6 F 9/38

技術表示箇所

3 1 0 X

審査請求 未請求 請求項の数20 OL (全 17 頁)

(21)出願番号 特願平9-52772

(22)出願日 平成9年(1997)3月7日

(31)優先権主張番号 特願平8-60571

(32)優先日 平8(1996)3月18日

(33)優先権主張国 日本(JP)

(71)出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72)発明者 海永 正博

東京都小平市上水本町五丁目20番1号 株

式会社日立製作所半導体事業部内

(72)発明者 斎藤 靖彦

東京都小平市上水本町五丁目20番1号 株

式会社日立製作所半導体事業部内

(74)代理人 弁理士 小川 勝男

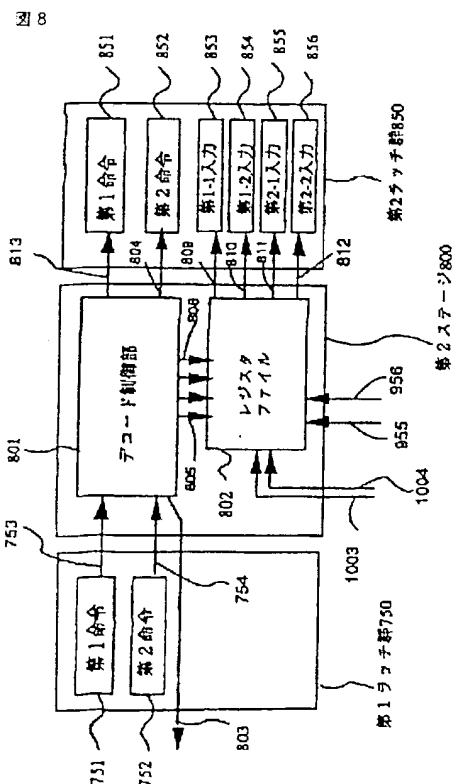
(54)【発明の名称】 データ処理装置

(57)【要約】

【課題】 スーバスカラ方式におけるデータ・ハザードのよるパイプライン・ストールを削減し、処理速度の向上を実現することにある。

【解決手段】 隣接する2つの2オペランド命令が、1つの3オペランド命令と同等であることを検出する回路と、そうであれば2つの命令を1つの3オペランド命令に統合して後続の実行ステージに送出する回路を命令デコードに設ける。また隣接する2つの命令がデータフローの関係にあるが1つの3オペランド命令には統合できないことを検出すると、先行命令のソースデータを後続命令のための演算器に送る回路を設ける。

【効果】 隣接命令間のデータフローにより従来であれば2クロックの時間を要していた2つの命令処理を1クロックで実行できる。したがって、全体としての実行クロック数を削減できる。



【特許請求の範囲】

【請求項1】複数のステージに分割して命令を実行するデータ処理装置であって、

前記複数のステージは、少なくとも命令メモリから命令を取り込む第1のステージと、前記第1のステージで取り込んだ命令を解読する第2のステージと、前記第2のステージで解読された命令を実行する第3のステージと、前記第3のステージで実行された結果をレジスタに書き込む第4のステージとであり、

前記命令メモリに格納される第1の命令フォーマットの命令を第2の命令フォーマットの命令に変更して実行することを特徴とするデータ処理装置。

【請求項2】前記第1の命令フォーマットは、演算命令において第1のオペランドと第2のオペランドとを演算し、第2のオペランドに演算結果を格納する命令フォーマットであり、

前記第2の命令フォーマットは、演算命令において第1のオペランドと第2のオペランドとを演算し、第3のオペランドに演算結果を格納する命令フォーマットであることを特徴とする請求項1に記載のデータ処理装置。

【請求項3】前記第2のステージは、先行命令がレジスタ間のデータ転送命令であることを検出し、後続命令が演算命令であることを検出し、さらに先行命令の転送先レジスタ番号と後続命令の転送先レジスタ番号が同一であることを検出して、

前記第2の命令フォーマットの演算命令に変換して前記第3のステージに送出することを特徴とする請求項2に記載のデータ処理装置。

【請求項4】前記請求項3に記載のデータ処理装置は、単一の半導体基板上に形成される。

【請求項5】前記先行命令が転送元レジスタの内容をそのまま転送先レジスタに転送するデータ転送命令である請求項4に記載のデータ処理装置。

【請求項6】前記先行命令が転送先のレジスタの内容をシフトして転送先のレジスタに転送するデータ転送命令である請求項4に記載のデータ処理装置。

【請求項7】前記先行命令が転送元レジスタの内容を0拡張または符号拡張して転送元のレジスタに転送するデータ転送命令である請求項4に記載のデータ処理装置。

【請求項8】前記第2の命令フォーマットは、前記第1の命令フォーマットの命令を複数組み合わせさせた命令を有することを特徴とする請求項1に記載のデータ処理装置。

【請求項9】前記第2のステージは、先行命令がレジスタ間のデータ転送であることを検出し、後続命令が固定ビットシフト命令であることを検出し、さらに先行命令の転送先レジスタ番号と後続命令の転送元レジスタ番号が同一であることを検出して、前記第2の命令フォーマットである1つのシフト命令に変換して第3のステージに送出することを特徴とする請

求項8に記載のデータ処理装置。

【請求項10】前記第2ステージは、先行命令がレジスタ間のデータ転送命令であることを検出し、後続命令が演算命令であることを検出し、さらに先行命令の転送先レジスタ番号と後続命令の転送元レジスタ番号が同一であることを検出して、

後続命令を先行命令とデータフローの関係にない前記第2の命令フォーマットの演算命令に変換して前記第3のステージに送出し、複数の同ステージを並列実行可能とすることを特徴とする請求項2に記載のデータ処理装置。

【請求項11】前記第1の命令フォーマットは、2バイト固定長命令であることを特徴とする請求項10に記載のデータ処理装置。

【請求項12】前記先行命令が転送元レジスタの内容をそのまま転送先レジスタに転送するデータ転送命令である請求項11に記載のデータ処理装置。

【請求項13】前記先行命令が転送先のレジスタの内容をシフトして転送先のレジスタに転送するデータ転送命令である請求項11に記載のデータ処理装置。

【請求項14】前記先行命令が転送元レジスタの内容を0拡張または符号拡張して転送元のレジスタに転送するデータ転送命令である請求項11に記載のデータ処理装置。

【請求項15】パイプライン方式のデータ処理装置であって、

命令メモリに格納される固定長命令を読み込む第1のステージと、

読み込まれた複数の命令が実行するデータに依存性が有り、かつ前記複数の命令に所定の関係がある場合、前記複数の命令を複数のパイプラインで並列に実行できるように前記複数の命令を変更する第2のステージと、変更された前記複数の命令を並列に実行する第3のステージとを有することを特徴とするデータ処理装置。

【請求項16】請求項15に記載の第1のステージは2つの命令を同時に読み込み、第2のステージは2つの命令を2本のパイプラインで並列に実行できるように前記2つの命令を変更することを特徴とするデータ処理装置。

【請求項17】請求項16に記載の第1のステージは、2バイト固定長命令を読み込むことを特徴とするデータ処理装置。

【請求項18】CPUと命令メモリとを単一の半導体基板上に形成するマイクロコンピュータであって、

前記CPUは、命令メモリに格納される2バイト固定長命令を2つ読み込む命令フェッチユニットと、読み込まれた前記2つの命令が実行するデータに依存性が有り、かつ前記2つの命令に所定の関係がある場合、2つの命令を2本のパイプラインで並列に実行できるよ

うに前記2つの命令を変更する命令デコーダと、変更された2つの命令を並列に実行する2つの4バイト長の演算器とを有することを特徴とするマイクロコンピュータ。

【請求項19】請求項18に記載の前記命令デコーダは、演算命令において第1のオペランドと第2のオペランドとを演算し、

第2のオペランドに演算結果を格納する命令を、第1のオペランドと第2のオペランドとを演算し、第3のオペランドに演算結果を格納する命令に変更することを特徴とするマイクロコンピュータ。

【請求項20】請求項18に記載の前記命令デコーダは、先行命令がレジスタ間のデータ転送命令であることを検出し、後続命令が演算命令であることを検出し、さらに先行命令の転送先レジスタ番号と後続命令の転送元レジスタ番号が同一であることを検出して、後続命令を先行命令とデータフローの関係にない演算命令に変更することを特徴とするマイクロコンピュータ。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、マイクロプロセッサやマイクロコンピュータ等のデータ処理装置に係わり、特にスーバスカラ等の並列処理を行うデータ処理装置に適用して有効な技術に関する。

【0002】

【従来の技術】マイクロプロセッサ(CPU(Central Processing Unit)、マイクロコンピュータ等の総称として以下使用する。)は、命令の列を順次にフェッチし、解読し、実行していく。マイクロプロセッサが実行していく命令は、デコード回路の単純化を狙って現在固定長のものが広まってきている。固定長命令をパイプライン方式(Pipelining)で実行するマイクロプロセッサは、RISC(Reduced Instruction Set Computer)型プロセッサと呼ばれている。

【0003】図1は、マイクロプロセッサのパイプライン化された実現方法を示したものである。ここでは簡単化のため通常は存在するメモリアクセスのステージ(MEM)を省略してある。個別のステージ(101、103、105、107)は1単位の時間刻み(クロック)で実行され、最初のステージから最後のステージまでラッチ群(102、104、106)を介して順次に処理を積み重ねていくことで個別の命令処理が完了する。第1ステージ101は命令フェッチを行う(IF)。第2ステージ103は命令の解釈及びレジスタの読み出しを行う(ID)。第3ステージ105は命令機能が指定した演算を実行する(EX)。第4ステージ107は演算結果を信号線108を介して第2ステージ103内に配置されたレジスタに書き込みを行う(WB)。

【0004】図2にはパイプラインで4つの命令を処理していく時の概念図が示される。後続の命令が先行命令

のレジスタの内容を使用する場合は、後続の命令のパイプラインに空きができてしまう(データ・ハザードによるパイプライン・ストールと呼ばれる)。この様子が図2の(a)に示されている。図2の(a)内の左下を向いた2つの矢印は先行命令のレジスタ書き込み後、後続命令のレジスタ読み出しを示している。

【0005】したがって、この問題を解決する手段として、後続の命令が前の演算結果を使用する場合にはその値を信号線108を介して第3ステージ105内の演算器にも送出する。以上のための制御線が信号線109、110である。この調整はフォワーディング(Forwarding)として知られており、これにより1クロック毎の実行が可能となる。なお、図2の(b)内の左下を向いた2つの矢印はフォワーディングを示している。したがって個別の命令処理に要するクロック数は例えば4となる。しかし、個別のステージが毎クロック新たな命令を処理していくので、命令処理は1クロック当たり1命令となる。したがって、1命令が1クロックで実行できるので、ある処理(プログラム)を行うための実行命令数が少ないほど実行時間が短くなる。

【0006】なお、パイプライン及びフォワーディングについては、1994年Morgan Kaufman Publishers, Inc. 発行のHennessy et al. 「Computer Organization and Design」第6章Enhancing Performance with Pipelining(362頁から450頁)に記載されている。

【0007】次に、マイクロプロセッサの処理速度を向上する方式の1例として、スーバスカラ方式(Superscalar)がある。スーバスカラ方式は、同時に実行できる演算器の数を複数個、例えば2個にし、それに応じて命令フェッチと命令デコードも1時期に2つ行えるようにしたものである。この場合、図3の(a)データ依存無しに示されるように、理想的には1クロック毎に2つの命令が実行可能にされるので、通常のパイプライン方式に較べ実行時間が半になる。なお、スーバスカラ方式については、日経エレクトロニクス、1989年11月27日号(No.487)、191頁から200頁の「次世代RISC、並列処理を導入しCMOSで100MIPSせらう」に記載されている。

【0008】従来のスーバスカラ方式を採用しているRISC型のマイクロプロセッサの命令長は4バイト固定であり、算術演算等の演算命令のオペランド数は3つとなっているのが一般的である。この例は、特開平2-130634号に記載されている。一方、コード効率を高める(命令を格納するメモリの使用量を少なくする)ために、2バイト固定長命令のRISC型のマイクロプロセッサがある。ただし、前記2バイト固定長命令のRISC型のマイクロプロセッサにはスーバスカラ方式は採用されていない。この例は、特開平5-197546号に記載されている。

【0009】

【発明が解決しようとする課題】スーパースカラ方式により生ずる課題を明らかにするために、図3を用いて説明する。図3に示される命令の動作が下記に示される。

【0010】(1) mov R3, R2 「レジスタR3の内容をレジスタR2に複写」

(2) mov #32, R5 「データ*32*をレジスタR5に複写」

(3) add R4, R2 「レジスタR4の内容とR2の内容を加算して、結果をR2に格納」

(4) and R3, R5 「レジスタR3の内容とR5の内容を論理積して、結果をR5に格納」

上記命令(1)と命令(2)、及び命令(3)と命令

(4)にはそれぞれデータの依存性(データフロー)はない。しかし、命令(1)と命令(3)、及び命令

(2)と命令(4)にはそれぞれデータの依存性(データフロー)がある。すなわち、命令(1)と命令(3)の両方でレジスタR2を使用する。また、命令(2)と命令(4)の両方でレジスタR5を使用する。従って、命令(1)の実行後に命令(3)を実行しなければならない。また、命令(2)の実行後に命令(4)を実行しなければならない。

【0011】すなわち、同時に実行する命令間にデータ依存性が無い場合、図3の(a)に示されるようにパイプラインの空きが無く、2命令が完全に並列実行されるので、従来の同時に1命令のみを実行する場合の2倍の処理速度が得られる。しかし、同時に実行する命令間にデータ依存性がある場合、図3の(b)に示されるように、パイプラインに乱れが出てしまい、従来の同時に1命令のみを実行する場合と同一の処理速度になってしまう。

【0012】そのために、図3の(c)に示されるように、同時に実行する命令間にデータ依存性がある場合、後続命令は次のパイプラインに回し、後続命令の替りに無処理命令nopを先行命令と同時に実行して、パイプラインの乱れを回避する方法が考えられる。しかし、無駄な命令が増え、全体の実行命令数が増加して実行時間が長くなる。

【0013】次に命令フォーマット及び命令体系により生ずる課題を明らかにするため、図4及び図5を用いて以下に説明する。

【0014】図4には、4バイト・3オペランド命令(4バイト固定長命令)体系の場合の命令形式(命令フォーマット)と命令レパートリの例が示される。この図でOPフィールド401は命令機能を特定する。S1フィールド403は第1入力特定するレジスタ番号(第1オペランド)、S2フィールド404は第2入力特定

$$a=b+c+d;$$

これを4バイト・3オペランド命令体系の命令列(命令列(A1))に変換すると以下ようになる。

【0019】

add Rb, Rc, Ra

定するレジスタ番号(第2オペランド)、Dフィールド402は出力を特定するレジスタ番号(第3オペランド)が置かれている。すなわち、この命令形式は3つのオペランドを指定することができる。命令機能には、複写(データ転送)、加算、減算などがある。さらに、4バイト命令体系の命令長の余裕から1ビット左シフト加算命令aslladdや0拡張加算命令zextaddなどの複合命令も提供される。aslladd命令は第1オペランドのビットパターンを1ビット左シフトした後で通常に加算を行うもので、zextadd命令は第1オペランドのビットパターンの左半分を0にした後で通常に加算を行うものである。なおここでは簡単化のため通常は存在するであろうメモリアクセス命令や分岐命令等を省略してある。なお複写命令(データ転送命令)の場合S2フィールド404は無視され、S1フィールド403で特定されたレジスタ(転送元レジスタ)内容がそのままDフィールド402で特定されたレジスタ(転送先レジスタ)に複写(転送)される。

【0015】図5には、2バイト・2オペランド命令(2バイト固定長命令)体系の場合の命令形式と命令レパートリの例が示される。図5でOPフィールド501は命令機能を特定する。S1フィールド503は第1入力特定するレジスタ番号(第1オペランド)、Dフィールド502は第2入力特定するレジスタ番号(出力を特定するレジスタ番号と同一、第2オペランド)が置かれている。すなわち、この命令形式は2つのオペランドを指定することができる。図4と較べてFS2フィールドが存在しない点が図4の命令形式と明確に異なっている部分である。すなわち、オペランドの数が1つ少ない。さらに残りのフィールド長も図4のものに較べて短くなっている。

【0016】命令機能には1入力転送命令として複写命令(データ転送命令)、0拡張命令、符号拡張命令、1ビット左シフト命令、2入力演算命令として加算命令、減算命令等がある。このうち1ビット左シフト命令は、命令長の都合で入力レジスタ(転送元レジスタ)と出力レジスタ(転送先レジスタ)の番号が同じである。したがってこの場合、S1フィールドはレジスタ番号でなく、asll命令を特定するための拡張命令コードが格納される。

【0017】さて、4バイト・3オペランド命令体系と2バイト・2オペランド命令体系の利害得失を明確化するために例えば、以下の式を考える。

【0018】

(A)

add Ra, Rd, Ra

一方これを2バイト・2オペランド命令体系の命令列(命令列(A2))に変換すると以下ようになる。

【0020】

```
mov    Rb, Ra
add     Rc, Ra
add     Rd, Ra
```

4バイト・3オペランドの命令体系であれば、実行命令数は2であるが、命令メモリでの格納（および実行のための命令フェッチ）バイト数は8バイトである。一方2バイト・2オペランドの命令体系であると、実行命令数は3に増えるが、命令メモリでの格納（および実行のための命令フェッチ）バイト数は6バイトに減少する。この傾向は一般的に成立する。そして、4バイト・3オペランド命令体系は2バイト・2オペランド命令体系に比べ実行命令数が1～2割程度少ないが、格納バイト数は6割程度多くなる、という点が一般的に認められる。

【0021】しかし、2バイト・2オペランドの命令体系には1つ課題が存在する。それは2オペランド命令体系の場合に必要な余分なデータ転送命令にかかわる。上の式（A）でも同様に説明できるのであるが、ここでは以下の式（B）で説明する。

【0022】 $a=b+c$;

これを4バイト・3オペランドの命令体系の命令列（命令列（B1））に変換すると以下ようになる。

【0023】

```
add    Rb, Rc, Ra
```

一方これを2バイト・2オペランドの命令列（命令列（B2））に変換すると以下ようになる。

【0024】

```
mov    Rb, Ra
add     Rc, Ra
```

4バイト・3オペランドの命令体系であれば、パイプラインの片方だけを使用して1クロックで実行できる。一方2バイト・2オペランドの命令体系であれば、余分に必要なとなった複写（データ転送）命令movと後続の加算命令addの2つの命令間にデータフローが存在する。つまり先行命令の結果の値を後続命令が使用している。したがって先行命令movの結果を待って後続命令addを実行する必要があり、2クロックの実行時間がかかる。以下の命令列

```
mov    Rb, Ra
add     Rc, Rd
```

であれば、2つの命令間でデータフローがないので、2つのパイプラインを使用して1クロックで実行できる訳であるが、式（B）に対応する命令列（B2）ではデータフローが存在することにより処理時間が余分にかかることになる。スーバスカラ方式を採用した場合、2バイト・2オペランド命令体系は4バイト・3オペランド命令体系に比べ、実行命令数の多さ以上に実行時間がかかる傾向があるといえる。

【0025】なお、2バイト・2オペランド命令体系の課題を4バイト・3オペランド命令体系と比較して説明したが、4バイト・3オペランド命令体系においても、4オペランドの演算を実行する場合、前記命令列（A

1）のようにデータフローが存在し、2バイト・2オペランド命令体系と同様な課題が存在する。

【0026】従来から存在するマイクロプロセッサは、ソフトウェア資産の蓄積があり、これまで築き上げてきたソフトウェア資産の継承の関係で、命令フォーマット、命令体系を変更することは困難である。従って、従来の命令フォーマット、命令体系を維持したまま、処理速度の向上を図る必要がある。

【0027】本発明の課題は、スーバスカラ方式におけるデータ・ハザードのよるパイプライン・ストールを削減し、処理速度の向上を実現することにある。

【0028】本発明の他の課題は、実行命令数を削減し、処理速度の向上を実現することにある。

【0029】さらに、本発明の他の課題は、2バイト・2オペランド命令体系を実行するデータ処理装置の処理速度の向上を実現することにある。

【0030】本発明の前記並びにその他の課題と新規な特徴は本明細書の記述及び添付図面から明らかになるであろう。

【0031】

【課題を解決するための手段】本願において開示される発明のうち代表的なものの概要を簡単に説明すれば下記の通りである。

【0032】パイプライン方式のデータ処理装置は、命令メモリに格納される固定長命令を読み込むステージと、読み込まれた複数の命令が実行するデータに依存性があり、かつ前記複数の命令に所定の関係がある場合、前記複数の命令を複数のパイプラインで並列に実行できるように前記複数の命令を変更するステージと、変更された前記複数の命令を並列に実行するステージとを有する。

【0033】命令体系上は2バイト2オペランド命令体系であるが、内部処理的には3オペランド命令体系として処理する。つまり、命令フェッチステージは2命令をフェッチする。命令デコードステージは2つの隣接した命令をデコードする。演算ステージの演算器は2組用意する。そして、隣接する2つの2オペランド命令が、1つの3オペランド命令と同等であることを検出する手段と、そうであれば2つの命令を1つの3オペランド命令に統合して後続の実行ステージに送出する手段を命令デコードに設ける。これにより、1つの3オペランド命令として実行ステージに送られ1つのクロックで実行される。また隣接する2つの命令がデータフローの関係にあるが1つの3オペランド命令には統合できないことを検出すると、先行命令のソースデータを後続命令のための演算器に送る手段を設ける。

【0034】これにより、2つの命令を同時に実行することになる。以上の2つにより、隣接命令間のデータフローにより従来であれば2クロックの時間を要していた2つの命令処理を1クロックで実行できることにな

る。したがって、全体としての実行クロック数を削減できる。

【0035】

【発明の実施の形態】本発明の実施例に係るマイクロプロセッサを順次項目に従って説明する。

【0036】《マイクロプロセッサのパイプラインデータバス》図6には本発明の実施例に係るマイクロプロセッサのパイプラインのデータバスが示される。前記マイクロプロセッサは図5に示すような2バイト・2オペランド命令体系の命令をフェッチし、実行するものであるとして以下説明する。

【0037】第1ステージ700は命令フェッチステージである。第2ステージ800は命令デコードステージである。第3ステージ900は演算ステージである。第4ステージ1000はレジスタへの書き込みとフォワーディングを行うステージである。前記各ステージの間には、第1ラッチ群750、第2ラッチ群850及び第3ラッチ群950がある。なお、図6以下の実施例における各ステージは、データの流れを示すもので、各ステージ内に記載される回路等の物理的な配置を示すものではない。

【0038】《命令フェッチステージ》図7には第1ステージ700と第1ラッチ群750との詳細ブロック図が示される。第1ステージ700は、プログラムカウンタ(PC)701とフェッチ制御部702と命令メモリ703とで構成される。第1ステージ700の命令フェッチステージの役割は命令メモリ内の命令を次の第2ステージ800の命令デコードステージに渡すことである。

【0039】プログラムカウンタ701の指すアドレスを信号線704に送出し命令メモリ703内の命令4バイト(2命令)を信号線705を介してフェッチ制御部702にフェッチする。フェッチ制御部702にフェッチされた2つの命令を信号線803に従って、信号線706、707に送出する。それから第1ラッチ群750内のラッチ751に信号線706の内容が格納され、ラッチ752に信号線707の内容が格納される。ラッチ751には第1命令が、ラッチ752には第2命令が格納される。ここで、命令列の中において第1命令は第2命令よりも先にある。なお、本願では第1命令を先行命令、第2命令を後続命令ともいう。

【0040】また、プログラムカウンタ701の値に4を加えた値をプログラムカウンタ701に設定しなおす。プログラムカウンタ701の値(命令メモリをアクセスするアドレスの値)は2の倍数という制約のもとで命令メモリから4バイト分の命令(2命令)をフェッチし第1ラッチ群750内にラッチするよう第1ステージ700が動作する。但し、常に命令メモリからフェッチした4バイト分の命令をそのまま第1ラッチ群750にラッチするものではない。すなわち、第2ステージ800

0である命令デコードステージから見て、次に欲しい命令が現命令の何バイト先かの情報を信号線803を介して第1ステージ700のフェッチ制御部702に送る。第1ステージ700のフェッチ制御部702はそれに応じてフェッチ制御部702内に存在するバッファを活用して命令デコードステージの望みの4バイト(2命令)を信号線706、707に送出し、第1ラッチ群750内のラッチ751、752に格納する。

【0041】《命令デコードステージ》図8には第2ステージ800と第2ラッチ群850との詳細ブロック図が示される。第2ステージ800は、デコード制御部801とレジスタファイル802とで構成される。第2ステージ800の命令デコードステージの役割は以下の通りである。

(1) 2つの命令で使用される入力データを用意し、次の演算ステージ(第3ステージ900)に渡す。

【0042】(2) 2つの命令間のデータフローを検査し、先行命令(第1命令)の実行結果を後続命令(第2命令)が使用していなければ、2つの命令処理を演算ステージに依頼する。

【0043】(3) 2つの命令間のデータフローを検査し、先行命令の実行結果を後続命令が使用していれば、所定の規則に従い2つの命令を変更する。

【0044】(4) 演算ステージに処理依頼した命令数を命令フェッチステージに連絡し、次のパイプラインの処理に備える。

【0045】命令デコードステージ(第2ステージ800)の動作を以下に説明する。図12にはデコード制御部801の一部の詳細ブロック図が示される。デコード制御部801はデータフロー検出回路DFDC、命令変換回路INCC等を有する。命令変換回路INCCは、セレクトSEL1から4を有し、データフロー検出回路DFDCの制御に基づいてラッチ751、752の内容を加工し、ラッチ851、852の内容に変換する。

【0046】ラッチ751の内容である第1命令のOPフィールドをOP-1、DフィールドをD-1、S1フィールドをS1-1とする。ラッチ752の内容である第2命令のOPフィールドをOP-2、DフィールドをD-2、S1フィールドをS1-2とする。ラッチ851の内容である第1命令のOPフィールドをOPN-1、DフィールドをDN-1、S1フィールドをS1N-1とする。ラッチ852の内容である第2命令のOPフィールドをOPN-2、DフィールドをDN-2、S1フィールドをS1N-2とする。ラッチ852の内容である第2命令はさらにS2フィールドを有し、これをS2N-2とする。

【0047】デコード制御部801はラッチ群750内のラッチ751、752より先行命令と後続命令の2つの命令を信号線753、754を介して取り込む。そし

て先行命令のDフィールド(D-1)のレジスタ番号が後続命令のS1フィールド(S1-2)又はDフィールド(D-2)のレジスタ番号と等しいか否かをデータフロー検出回路DFDCで検査する。

【0048】レジスタ番号が等しくない場合、データフローは存在しないと判断できる。レジスタ番号が等しい場合、データフローが存在すると判断できる。そうすると、データフロー検出回路DFDCは、制御信号821から824を出力し、セクタSEL1から4をそれぞれ切り替えて信号線813、804を介して、ラッチ851、852に変換した第1命令、第2命令を格納する。なお、セクタSEL1、SEL2の一つの入力にはINCCで生成された無効命令NOP820が常時入力される。

【0049】さらに、セクタSEL2には、信号線840を介してデータフロー検出回路DFDCにより生成した新たな命令が入力される。信号線840によりセクタSEL2に入力される新たな命令は、データフロー検出回路DFDCがラッチ751のOP-1とラッチ752のOP-2に基づいて生成したものであり、ラッチ852のOP-2に格納される。生成される新たな命令の一例としては、OP-1が1ビットシフト命令asllでOP-2が加算命令addのときに生成される1ビットシフト加算命令aslladdがある。

【0050】セクタSEL3は、S1-1またはD-2の一方の値を選択し、S1N-2に格納するためのものである。

【0051】セクタSEL4は、S1-1またはS1-2の一方の値を選択し、S2N-2に格納するためのものである。

【0052】図11には命令デコードステージの2つの命令を演算ステージの2つの命令に変換する規則(条件と演算ステージに渡る命令)が示されている。第1命令は、無効命令nopに変換されるか又は変換されないかのどちらかである。第2命令は命令形式を図5の2バイト・2オペランド形式ものから図4の4バイト・3オペランド形式ものに変換されるか又は無効命令nopに変換される。図11のALUは算術演算(加算、減算等)や論理演算(論理積、論理和等)などの2入力演算命令を総称する命令名である。前述したように、zextALUは演算器への第1入力を0拡張し、そしてALU演算する命令である。asllALUは演算器への第1入力を1ビット左シフトし、そしてALU演算する命令である。

【0053】図11の(1)は2オペランド形式の演算命令で3オペランドの演算命令を実行するためには複写命令movと演算命令ALUとの2命令必要であったものを1つの3オペランドの演算命令ALUに変換するものである。複写命令movのDフィールドのレジスタ番号と演算命令ALUのDフィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第1命令が無効

命令nopに、第2命令が3オペランドの演算命令に変換されて渡される。

【0054】ラッチ851、852の各フィールドに格納される値を要約すると以下になる。なお、「←」は、「←」の右側の値を「←」の左側に格納することを意味する。

【0055】

```
IF (D-1) = (D-2),  
THEN OPN-1 ← nop,  
      OPN-2 ← OP-2,  
      DN-2 ← D-2,  
      S1N-2 ← S1-1,  
      S2N-2 ← S1-2
```

具体的には以下になる。ラッチ751のOP-1には「mov」が、D-1には「RN」が、S1-1には「Rm」が格納されているとする。また、ラッチ752のOP-2には「ALU」が、D-2には「RN」が、S1-2には「R1」が格納されているとする。ここでD-1とD-2が共に「RN」でレジスタ番号が一致することをデータフロー検出回路DFDCが検出する。するとデータフロー検出回路DFDCは、SEL1がnop命令820を選択するように821を介しセクタSEL1を制御し、nop命令820をラッチ851のOPN-1に格納する。データフロー検出回路DFDCは、ラッチ751のD-1、S1-1をそのまま信号線753、813を介してラッチ851のDN-1、S1N-1に格納する。

【0056】またデータフロー検出回路DFDCは、セクタSEL2がラッチ752のOP-2を選択するように制御信号822を介してセクタSEL2を制御し、ラッチ752のOP-2をラッチ852のOPN-2に格納する。さらにデータフロー検出回路DFDCは、セクタSEL3がラッチ751のS1-1を選択するように制御信号823を介しセクタSEL3を制御し、ラッチ751のS1-1をラッチ852のS1N-2に格納する。またデータフロー検出回路DFDCはラッチ752のD-2を信号線754を介してそのままラッチ852のDN-2に格納する。さらにデータフロー検出回路DFDCは、セクタSEL4がラッチ752のS1-1を選択するように834を介してセクタSEL4を制御し、ラッチ752のS1-1をS2N-2に格納する。

【0057】図11の(2)は複写命令movのDフィールドのレジスタ番号と演算命令ALUのS1フィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第1命令はそのまま、第2命令が3オペランドの演算命令に変換されて渡される。

【0058】ラッチ851、852の各フィールドに格納される値を要約すると以下になる。

```

IF (D-1) = (S1-2),
  THEN OPN-1 ← OP-1,
        DN-1 ← D-1,
        S1N-1 ← S1-1,
        OPN-2 ← OP-2,
        DN-2 ← D-2,
        S1N-2 ← S1-1,
        S2N-2 ← D-2

```

具体的には以下のようなものである。ラッチ751のOP-1には「mov」が、D-1には「RN」が、S1-1には「Rm」が格納されているとする。また、ラッチ752のOP-2には「ALU」が、D-2には「Rx」が、S1-2には「RN」が格納されているとする。ここでD-1とS1-2が共に「RN」でレジスタ番号が一致することをデータフロー検出回路DFDCが検出する。そしてデータフロー検出回路DFDCは、セクタSEL1がラッチ751のOP-1(この場合mov命令)を選択するように821を介しセクタSEL1を制御し、mov命令をラッチ851のOPN-1に格納する。

【0059】データフロー検出回路DFDCは、ラッチ751のD-1、S1-1を、そのまま信号線753、813を介してラッチ851のDN-1、S1N-1に格納する。またデータフロー検出回路DFDCは、セクタSEL2がラッチ752のOP-2を選択するように制御信号822を介してセクタSEL2を制御し、ラッチ752のOP-2をラッチ852のOPN-2に格納する。なお データフロー検出回路DFDCは、ラッチ752のD-2をそのまま信号線754、804を介してラッチ852のDN-2に格納する。さらにデータフロー検出回路DFDCは、セクタSEL3がラッチ751のS1-1を選択するように制御信号823を介してセクタSEL3を制御し、信号線804を介してラッチ852のS1N-2にラッチ751のS1-1を格納する。なおデータフロー検出回路DFDCは、ラッチ752のS1-2を信号線754、804を介して、そのままラッチ852のS2N-2に格納する。

【0060】なお、ラッチ851、852に具体的に格納される値を作っていく以上のような説明は図11の(2)以降では省略する。図11の(1)、(2)と同様なやり方でラッチ851、852に格納する値を作っていくからである。

【0061】図11の(3)は1オペランド形式の1ビット左シフト命令を2オペランド形式の1ビット左シフト命令に変換するものである。複写命令movのDフィールドのレジスタ番号と1ビット左シフト命令asllのDフィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第1命令が無効命令nopに、第2命令が2オペランドの1ビット左シフト命令asllに変換されて渡される。

【0062】すなわち、各フィールドは下記のように変

換される。

```

【0063】
IF (D-1) = (S1-2),
  THEN OPN-1 ← nop,
        OPN-2 ← OP-2,
        DN-2 ← D-2 or D-1,
        S1N-2 ← S1-1,
        S2N-2 ← NA

```

図11の(4)は第1命令が複写命令movで、第2命令又は条件が図11の(1)、(2)、(3)に該当しなかった場合である。この場合、演算ステージには第1命令はそのままで、第2命令が無効命令nopに変換されて渡される。「その他」の命令は1クロックずれた次のパイプラインで実行される。

【0064】すなわち、各フィールドは下記のように変換される。

```

【0065】
OPN-1 ← OP-1,
DN-1 ← D-1,
S1N-1 ← S1-1,
OPN-2 ← nop

```

図11の(5)は0拡張命令zextと演算命令ALUとを0拡張演算命令zextALUに複合するものである。0拡張命令zextのDフィールドのレジスタ番号と演算命令ALUのDフィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第1命令が無効命令nopに、第2命令が3オペランドの0拡張演算命令zextALUに変換されて渡される。

【0066】すなわち、各フィールドは下記のように変換される。

```

【0067】
IF (D-1) = (D-2),
  THEN OPN-1 ← nop,
        OPN-2 ← zextALU,
        DN-2 ← D-2 or D-1,
        S1N-2 ← S1-1,
        S2N-2 ← S1-2

```

図11の(6)は0拡張命令zextのDフィールドのレジスタ番号と加算命令addのS1フィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第1命令はそのままで、第2命令が3オペランドの0拡張加算命令zextaddに変換されて渡される。

【0068】すなわち、各フィールドは下記のように変換される。

```

【0069】

```

```

IF (D-1) = (S1-2) ,
THEN OPN-1 ← OP-1,
      DN-1 ← D-1,
      S1N-1 ← S1-1,
      OPN-2 ← zextadd,
      DN-2 ← D-2,
      S1N-2 ← S1-1,
      S2N-2 ← D-2

```

なお、加算命令add以外に可換な論理積命令andや論理和命令or等も同様な変換を行っても良い。

【0070】図11の(7)は第1命令が0拡張命令zextで、第2命令又は条件が図11の(5)又は(6)に該当しない場合である。この場合、演算ステージには第1命令はそのまま、第2命令が無効命令nopに変換されて渡される。「その他」の命令は1クロックずれた次のパイプラインで実行される。

【0071】すなわち、各フィールドは下記のように変換される。

```

【0072】
OPN-1 ← OP-1,
DN-1 ← D-1,
S1N-1 ← S1-1,
OPN-2 ← nop

```

図11の(8)は1ビット左シフト命令asllと演算命令ALUとを1ビット左シフト演算命令asllALUに複合するものである。1ビット左シフト命令asllのDフィールドのレジスタ番号と演算命令ALUのDフィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第1命令が無効命令nopに、第2命令が3オペランドの1ビット左シフト演算命令asllALUに変換されて渡される。

【0073】すなわち、各フィールドは下記のように変換される。

```

【0074】
IF (D-1) = (D-2) ,
THEN OPN-1 ← nop,
      OPN-2 ← asllALU,
      DN-2 ← D-2,
      S1N-2 ← S1-1,
      S2N-2 ← S1-2

```

図11の(9)は1ビット左シフト命令asllのDフィールドのレジスタ番号と加算命令addのS1フィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第1命令はそのまま、第2命令が3オペランドの1ビット左シフト加算命令aslladdに変換されて渡される。

【0075】すなわち、各フィールドは下記のように変換される。

【0076】

```

IF (D-1) = (S1-2) ,
THEN OPN-1 ← OP-1,
      DN-1 ← D-1,
      S1N-1 ← S1-1,
      OPN-2 ← aslladd,
      DN-2 ← D-2,
      S1N-2 ← S1-1,
      S2N-2 ← D-2

```

図11の(10)は第1命令が1ビット左シフト命令asllで、第2命令又は条件が図11の(8)又は(9)に該当しない場合である。この場合、演算ステージには第1命令はそのまま、第2命令が無効命令nopに変換されて渡される。「その他」の命令は1クロックずれた次のパイプラインで実行される。

【0077】すなわち、各フィールドは下記のように変換される。

```

【0078】
OPN-1 ← OP-1,
DN-1 ← D-1,
S1N-1 ← S1-1,
OPN-2 ← nop

```

図11の(11)は2つの命令間にデータフローがない場合のもので、命令の変換は行わない。

【0079】デコード制御部801で変換された新しい2つの命令は信号線813、804に送出され、それぞれ第2ラッチ群850内のラッチ851、852に格納される。また、データフロー検出回路DFDCにおける先行命令と後続命令との関係の検査結果を図11のPC更新の値に基づき命令フェッチステージ(第1ステージ700)に信号線803を介して連絡する。すなわち、次のパイプラインでデコードする2つの命令を指定する情報を命令フェッチステージに連絡する。

【0080】さらにデコード制御部801は先行命令のS1フィールド(S1-1)、Dフィールド(D-1)、さらに後続命令のS1フィールド503(S1-2)、Dフィールド502(D-2)の4つのレジスタ番号を信号線805、806、807、808を介してレジスタファイル802に送る。レジスタファイル802内の4つのレジスタの内容は、信号線809、810、811、812に読み出され、第2ラッチ群74内のラッチ853(第1-1入力)、ラッチ854(第1-2入力)、ラッチ855(第2-1入力)、ラッチ856(第2-2入力)に格納される。

【0081】図15には、レジスタファイル802のブロック図が示される。レジスタファイル802は、レジスタRGSTRとレジスタ制御回路RCCと等で構成される。レジスタRGSTRは、4本のリードポートと2本のライトポートとがあり、それぞれ信号線809、810、811、812、信号線955、956に接続される。従って、レジスタファイル802は4つのレジ

タの内容を同時に読み出すことができる。また、2つのレジスタに同時に書き込むことができる。

【0082】図11の(1)、(5)、(8)の場合は、(S1-1)と(S1-2)で指定される2つのレジスタの内容が信号線811、812に読み出され、ラッチ855(第2-1入力)、ラッチ856(第2-2入力)に格納される。

【0083】図11の(2)、(6)、(9)の場合は、(S1-1)で指定されるレジスタの内容が信号線809、811に読み出され、ラッチ853(第1-1入力)とラッチ855(第2-1入力)とに格納される。(D-2)で指定されるレジスタ内容が信号線812に読み出され、ラッチ856(第2-2入力)に格納される。

【0084】図11の(3)の場合は、(S1-1)で指定されるレジスタの内容が信号線811に読み出され、ラッチ855(第2-1入力)に格納される。

【0085】図11の(4)、(7)、(10)の場合は、(S1-1)で指定されるレジスタの内容が信号線809に読み出され、ラッチ853(第1-1入力)に格納される。

【0086】図11の(11)の場合は、(S1-1)、(D-1)、(S1-2)、(D-2)で指定される4つのレジスタの内容が信号線809、810、811、812に読み出され、ラッチ853(第1-1入力)、ラッチ854(第1-2入力)、ラッチ855(第2-1入力)、ラッチ856(第2-2入力)に格納される。

【0087】《実行ステージ》図9には第3ステージ900と第3ラッチ群950との詳細ブロック図が示される。第3ステージ900は、演算制御部901とALU(Alithmetic Logic Unit)等を含む演算器902、903と第1入力調整回路904、905、選択器906、907とで構成される。第3ステージ900である実行ステージの役割は、2つの命令の演算を実行することである。

【0088】演算器902と第1入力調整回路904は先行命令を演算するための回路で、第2ラッチ群850内の2つのラッチ853、854から第1-1入力、第1-2入力が信号線859、860を介して選択器906に送られる。また、第3ラッチ群950内の2つのラッチ953、954から第1出力、第2出力が信号線955、956を介して選択器906に送られる。

【0089】選択器906は信号線859、955及び956のうちの1つを信号線1001に従い選択して第1入力回路904及び信号線912を介して演算器902にデータを送る。また、選択器906は信号線860、955及び956のうちの1つを信号線1001に従い選択して信号線913を介して演算器902にデータを送る。

【0090】演算制御部901は第2ラッチ群850内のラッチ851の命令を取り込み、その命令機能に従い演算器902、第1入力調整回路904を信号線911と908で制御し、先行命令のための演算を行う。そして結果の値(第1出力)は第3ラッチ群950内のラッチ953に信号線918を介して格納される。

【0091】一方、演算器903と第1入力調整回路905は後続命令を演算するための回路で、第2ラッチ群850内の2つのラッチ855、856から第2-1入力、第2-2入力が信号線861、862を介して選択器907に送られる。また、第3ラッチ群950内の2つのラッチ953、954から第1出力、第2出力が信号線955、956を介して選択器907に送られる。

【0092】選択器907は信号線861、955及び956のうちの1つを信号線1002に従い選択して第1入力回路905及び信号線914を介して演算器903にデータを送る。また、選択器907は信号線862、955及び956のうちの1つを信号線1002に従い選択して信号線915を介して演算器903にデータを送る。演算制御部901は第2ラッチ群850内のラッチ852の命令を取り込み、その命令機能に従い演算器903、第1入力調整回路905を信号線910と909とで制御し、後続命令のための演算を行う。そして結果の値(第2出力)は第3ラッチ群950内のラッチ954に信号線919を介して格納される。

【0093】以上が、実行ステージ(第3ステージ900)の処理であるが、sladd命令やzextadd命令について補足説明しておく。aslladd命令やzextadd命令は、加算を実現できる演算器902または903への第1入力を微調整することで実現できる。すなわち第1入力を演算器に直接入力するのではなく、第1入力調整回路904または905に入力しそれを演算制御部901が制御し、1ビット左シフトや0拡張の調整を行っておき、それを演算器902または903へ入力し、そこで通常の加算をするよう制御することで実現できる。

【0094】《書き込みステージ》図10には第4ステージ1000の動作を説明するためのブロック図が示される。第4ステージ1000は、レジスタ番号解読回路1010とフォワーディング制御回路1020とで構成される。第4ステージ1000であるレジスタへの書き込みとフォワーディングを行うステージの役割は以下の通り。

【0095】(1) 2つの命令の演算結果を指定された番号のレジスタに書き込む。

【0096】(2) 2つの命令の演算結果が現クロックでの演算ステージ(次のパイプライン)で使用されるなら第2ラッチ群850内にラッチされている値でなく、第3ラッチ群950内にラッチされている値を演算器に入力するように調整する(フォワーディング)。

【0097】まず、(1)の処理から説明する。第4ス

ステージ1000は、第3ラッチ群950内のラッチ951、952から直前に演算された2つの命令を信号線957、958を介してレジスタ番号解読回路1010に取り込む。また第3ラッチ群950内のラッチ953、954から直前の演算結果の値を信号線955、956に送出する。そして、レジスタ番号解読回路1010は直前に実行された命令の2つのDフィールド内のレジスタ番号を信号線1003、1004に送出して第2ステージ800のレジスタファイル802の書き込みレジスタ番号を指定する。これで2つの演算結果の値がレジスタファイル802に書き込まれることになる。

【0098】次に(2)の処理を説明する。第4ステージ1000は、第2ラッチ群850内のラッチ851、852から今回演算すべき2つの命令を信号線857、858を介してフォワーディング制御回路1020に取り込む。また、第3ラッチ群950内のラッチ951、952から直前に演算された2つの命令を信号線957、958を介してフォワーディング制御回路1020に取り込む。そして、フォワーディング制御回路1020は直前に実行された命令の2つのDフィールド内のレジスタ番号と今回演算されるべき2つの命令のS1フィールド、S2フィールドの番号に同じものがあるか検査する。検査の結果同じものがあれば、その部分について、第2ラッチ群850内のラッチ853、854、855、856内の値でなく、第3ラッチ群950内のラッチ953、954内の値(信号線955、956)が演算器902、903に入力されるようにフォワーディング制御回路1020は信号線1001、1002を送出して2つの選択器906、907を制御する。

【0099】《命令列の処理》図13には本発明のスーパースカラ処理において命令列が個別のクロックでどのように処理されていくかが示されている。また、比較のため2命令が並列に実行できないときに無効命令nopを挿入するのみ場合に命令列が個別のクロックでどのように処理されていくかも示されている。本発明では、1クロック当たり2つの命令処理が可能となっている。また、本発明では、2命令が並列に実行できないときに無効命令nopを挿入する場合に比べて、実行命令数が6つ少なく実行時間が短くなる(この命令列においては約40%実行命令が少なくなる)。

【0100】先行命令がmov, zext, asll等の転送系命令で後続命令がaddなどの加算命令であれば、2つの命令を1つの命令に変換し、1つのクロックで実行するので、全体としてのクロック数を削減でき、高速化を図れる。また、先行命令が転送系命令で後続命令が演算命令であり、さらに両者間にデータフローが存在する場合でも、1つのクロックで実行するので、全体としてのクロック数を削減でき、高速化を図れる。

【0101】《マイクロコンピュータへの適用例》図14には本発明のスーパースカラ方式を用いたマイクロプロ

コンピュータシステムが示される。マイクロコンピュータMCUは、中央処理装置CPUと、浮動小数点処理ユニットFPUと、積和演算機能を有する乗算器MULTと、論理アドレスを物理アドレスに変換するメモリ管理ユニットMMUと、命令及びデータのキャッシュメモリCACHEと、キャッシュコントローラCCNTと、外部バスインタフェースEBIFと、32ビット論理アドレスバスLABUSと、32ビット物理アドレスデータバスPABUSと、32ビットデータバスDBUS、DBSとを単結晶シリコンのような半導体基板上に形成され、樹脂封止される(プラスチックパッケージに封止される)。

【0102】マイクロコンピュータMCUは、外部アドレスバスEABとデータバスEDBを介してDRAM等のダイナミック記憶素子をメモリセルに使用した半導体メモリ等からなる主記憶装置MMに接続される。

【0103】中央処理装置CPUは、図6に示されるパイプライン・データバスで構成される。ただし、第3ステージと第4ステージとの間にメモリアクセスステージを有し、いわゆる5段パイプラインを構成する。なお、データメモリと命令メモリ703は、キャッシュメモリCACHE又は主記憶装置MMに対応し、中央処理装置CPU内には存在しない。中央処理装置CPUは2バイト固定長命令の命令体系の命令を実行し、演算器902、903は、32ビット長のALU等をそれぞれ有する。また、レジスタファイル802は、32ビット長の汎用レジスタを16本を有する。すなわち、中央処理装置CPUは特開平5-197546号公報に記載される2バイト・2オペランド命令体系(命令セット)の命令を実行する。特開平5-197546号公報に記載されるCPUはスーパースカラ方式でない。これに比べて中央処理装置CPUはスーパースカラ方式であり、中央処理装置CPUは出願番号1992/897457号に記載される命令体系と同一の命令体系を実行できる。従って、従来のソフトウェアと互換性(オブジェクト・コード互換性)を維持しながら、高速性能を実現できる。また、2バイト固定長命令の特徴である高コード効率化も維持できる。

【0104】以上本発明者によってなされた発明を実施例に基づいて具体的に説明したが、本発明はそれに限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能であることはいうまでもない。例えば、図6以下の実施例では、2バイト・2オペランド命令体系の場合について説明したが、4バイト・3オペランド命令体系の場合にも適用できる。0拡張命令、0拡張演算命令について説明したが、符号拡張命令、符号拡張演算命令についても同様に適用できる。また、第1命令の転送命令のS1フィールドはレジスタを指定する場合について説明したが、即値データである場合についても適用できる。

【0105】

【発明の効果】本願において開示される発明のうち代表的なものによって得られる効果を簡単に説明すれば下記の通りである。

【0106】隣接命令間のデータフローを検出し、命令を変換することにより、並列に命令を実行できる。従って、従来であれば複数クロックの時間を要していた複数の命令処理を1クロックで実行できる。それによって、全体としての実行クロック数を削減できる。

【図面の簡単な説明】

【図1】マイクロプロセッサのパイプライン化された実現方式を示す図。

【図2】パイプライン処理の概念を示す。

【図3】スーパースカラ処理の概念を示す。

【図4】4バイト命令体系の命令形式と命令レパートリの例を示す。

【図5】2バイト命令体系の命令形式と命令レパートリの例を示す。

【図6】本発明の実施例に係るマイクロプロセッサのパイプラインのデータパスを示す図。

【図7】第1ステージと第1ラッチ群との詳細ブロック図。

【図8】第2ステージと第2ラッチ群との詳細ブロック図。

【図9】第3ステージと第3ラッチ群との詳細ブロック図。

【図10】第4ステージの動作を説明するブロック図。

【図11】命令デコードステージの2つの命令を演算ステージの2つの命令に変換する規則を示す。

【図12】デコード制御部の一部の詳細ブロック図を示す。

【図13】命令列が個別のクロックでどのように処理されていくかを示す。

【図14】本発明のスーパースカラ方式を用いたマイクロコンピュータシステムの図。

【図15】レジスタファイルのブロック図。

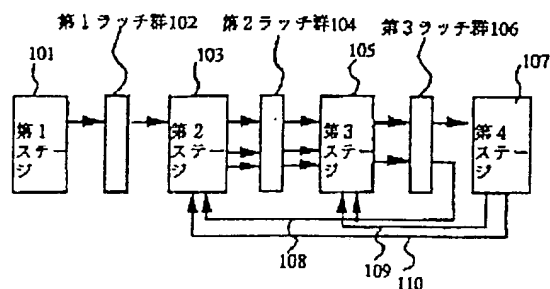
【符号の説明】

101……第1ステージ、103……第2ステージ、1

05……第3ステージ、107……第4ステージ、108、109、110……信号線、401……OPフィールド、402……Dフィールド、403……S1フィールド、404……S2フィールド、501……OPフィールド、502……Dフィールド、503……S1フィールド、700……第1ステージ、800……第2ステージ、900……第3ステージ、1000……第4ステージ、701……プログラムカウンタ、702……フェッチ制御部、703……命令メモリ、704、705、706、707……信号線、751、752……ラッチ、801……デコード制御部、802……レジスタファイル、803、804、805、806、807、808、809、810、811、812、813……信号線、851、852、853、854、855、856……ラッチ、857、858、859、860、861、862……信号線、901……演算制御部、902……演算器、903……演算器、904……第1入力調整回路、905……第1入力調整回路、906……選択器、907……選択器、908、909、910、911、912、913、914、915、916、917、918、919……信号線、951、952、953、954……ラッチ、955、956、957、958……信号線、1001、1002、1003、1004……信号線、1010……レジスタ番号制御回路、1020……フォワード制御回路、INCC……命令変換回路、DFDC……データフロー検出回路、MCU……マイクロコンピュータ、CPU……中央処理装置、FPU……浮動小数点処理ユニット、MULT……乗算器、MMU……メモリ管理ユニット、CACHE……命令及びデータのキャッシュメモリ、CCNT……キャッシュコントローラ、EBIF……外部バスインタフェース、LABUS……32ビット論理アドレスバス、PABUS……32ビット物理アドレスデータバス、DBUS、DBS……32ビットデータバス、EAB……外部アドレスバス、EDB……外部データバス、MM……主記憶装置、RCC……レジスタ制御回路、RGSTR……レジスタ。

【図1】

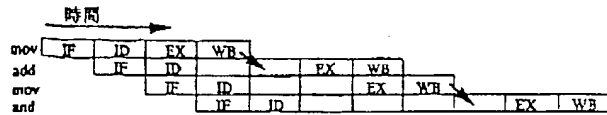
図1



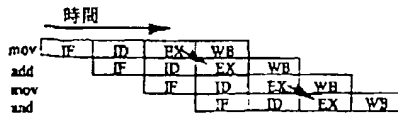
【図2】

図2

(a) データハザードによりパイプラインストールが発生した場合

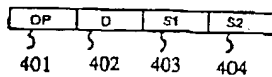


(b) フォーワーディングによりパイプラインストールがない場合



【図4】

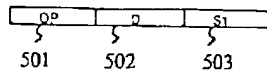
図4



複写	mov Rn,Rn	mov Rn Rn ***
加算	add Rn,R1,Rn	add Rn Rn R1
0拡張加算	zextadd Rn,R1,Rn	zextadd Rn Rn R1
符号拡張加算	sxtadd Rn,R1,Rn	sxtadd Rn Rn R1
1ビット左 シフト加算	aslladd Rn,R1,Rn	aslladd Rn Rn R1

【図5】

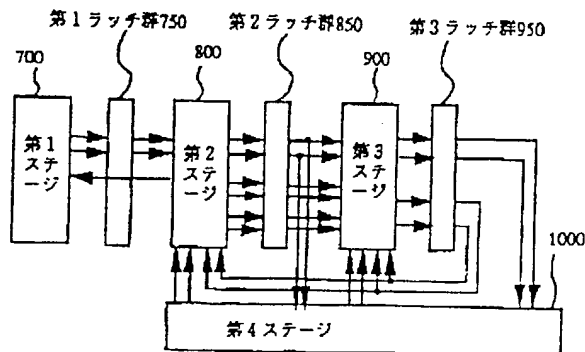
図5



複写命令	mov Rn,Rn	mov Rn Rn
加算命令	add Rn,Rn	add Rn Rn
0拡張命令	zext Rn,Rn	zext Rn Rn
符号拡張命令	sxt Rn,Rn	sxt Rn Rn
1ビット左 シフト命令	asll Rn,Rn	asll Rn ***

【図6】

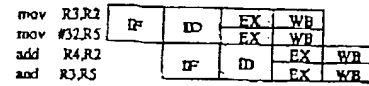
図6



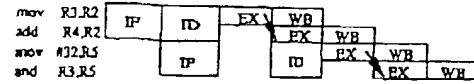
【図3】

図3

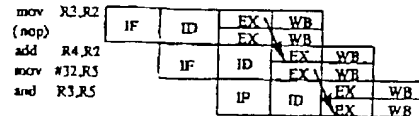
(a) データ依存無し



(b) データ依存有り

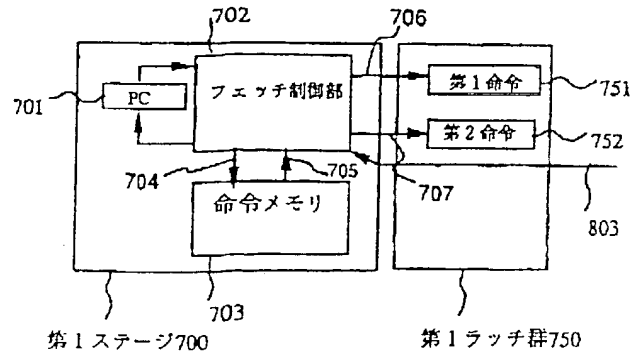


(c) データ依存有り (無効命令挿入)

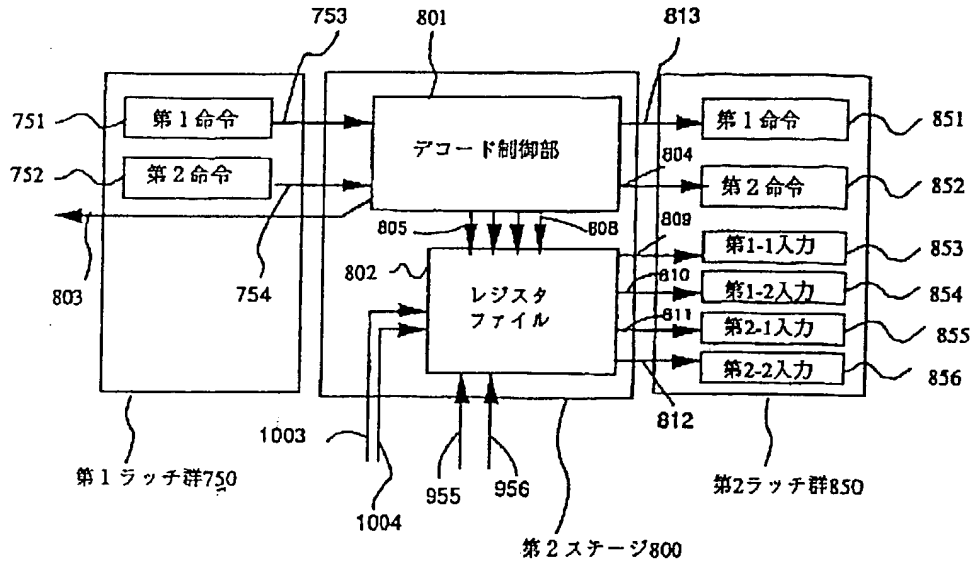


【図7】

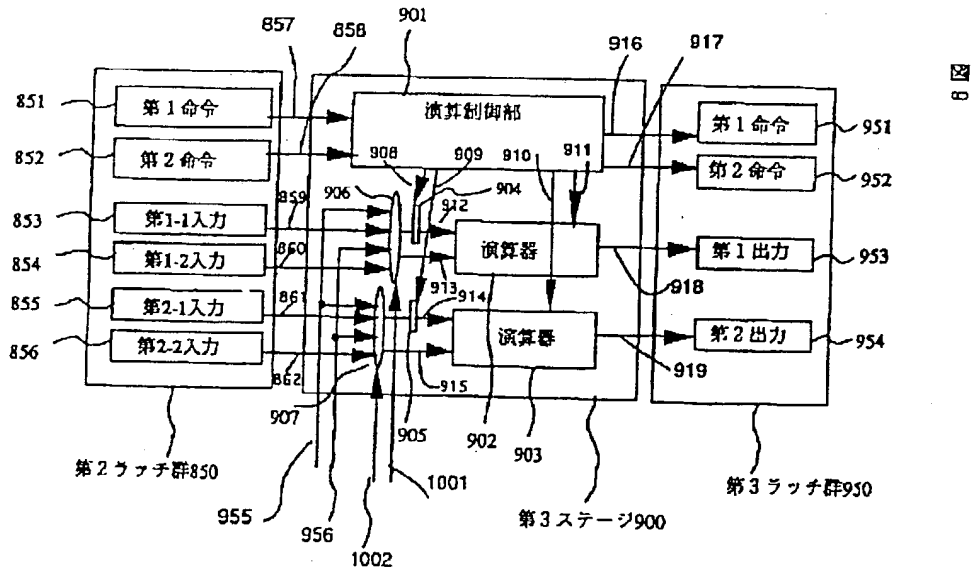
図7



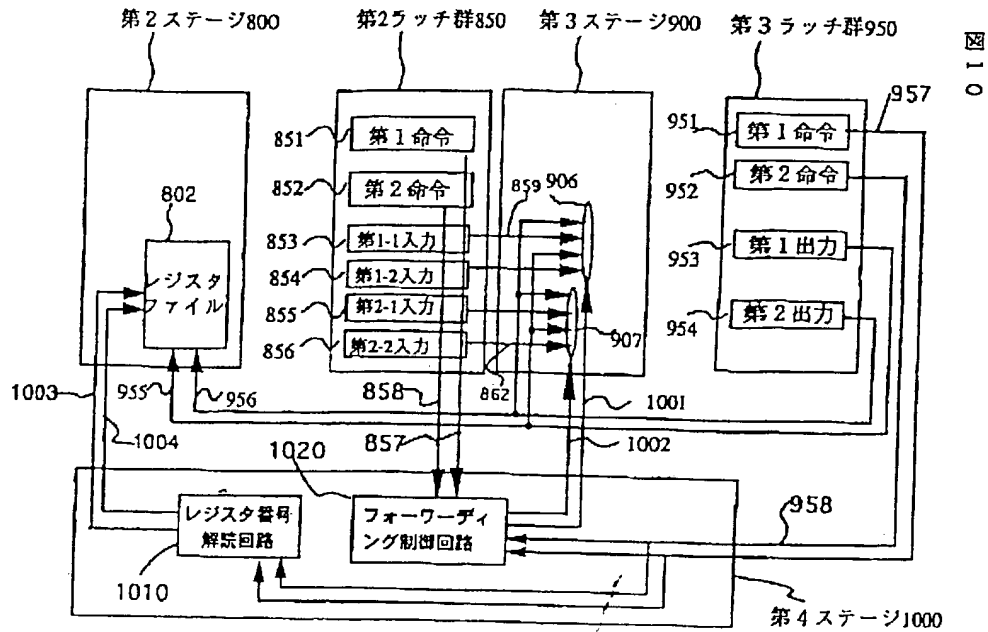
【図8】



【図9】



【図10】



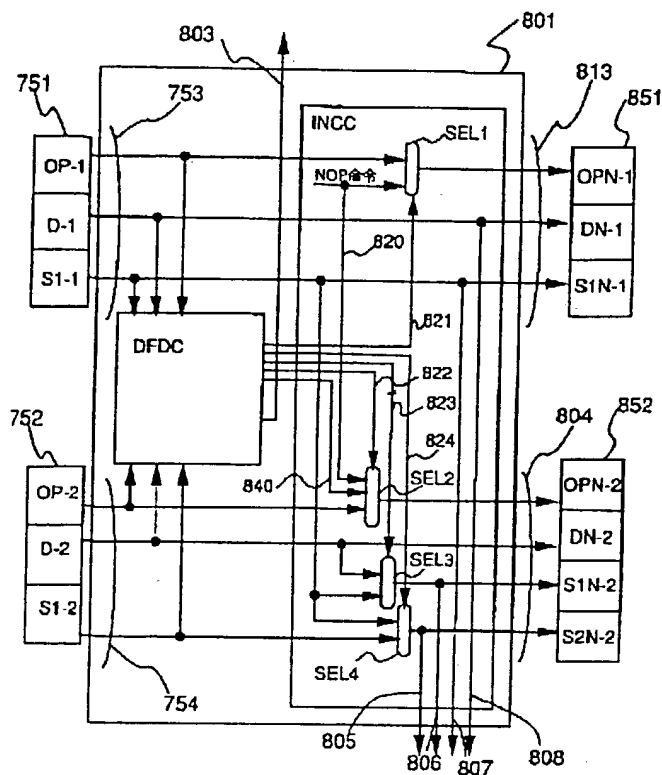
【図11】

	デコードステージへの命令				演算ステージへの命令	
	第1命令	第2命令	条件	PCの更新	第1命令	第2命令
(1)	mov Rm,Rn	ALU Rl,Rx	n=x	+4	nop	ALU Rm,Rl,Rn
(2)	mov Rm,Rn	ALU Rl,Rx	n=l	+4	mov Rm,Rn	ALU Rx,Rm,Rx
(3)	mov Rm,Rn	asl Rx,Rx	n=x	+4	nop	asl Rm,Rn
(4)	mov Rm,Rn	その他	---	+2	mov Rm,Rn	nop
(5)	zext Rm,Rn	ALU Rl,Rx	n=x	+4	nop	zextALU Rm,Rl,Rn
(6)	zext Rm,Rn	add Rl,Rx	n=l	+4	zext Rm,Rn	zextadd Rx,Rm,Rx
(7)	zext Rm,Rn	その他	---	+2	zext Rm,Rn	nop
(8)	asl Rm,Rn	ALU Rl,Rx	n=x	+4	nop	aslALU Rm,Rl,Rn
(9)	asl Rm,Rn	add Rl,Rx	n=l	+4	asl Rm,Rn	asladd Rm,Rx,Rx
(10)	asl Rm,Rn	その他	---	+2	asl Rm,Rn	nop
(11)	ALU Rm,Rn	ALU Rl,Rx	n=l n≠x	+4	ALU Rm,Rn	ALU Rl,Rx

図
11

【図12】

図12



【図13】

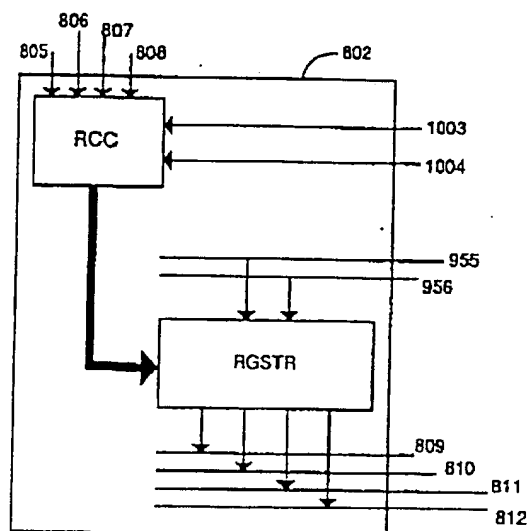
図13

クロック 第2ステージ800 第3ステージ900 nopを挿入した

への命令	への命令	場合
1 mov Re,Rf asll Rf,Rf		
2 zext Rb,Ra add Rc,Ra	nop asll Re,Rf	mov Re,Rf nop
3 mov Ra,Rb add Rc,Rb	nop zextadd Rb,Rc,Ra	asll Rf,Rf zext Rb,Ra
4 mov Rb,Rd add Rd,Re	nop add Ra,Rc,Rb	add Rc,Ra nop
5 add Rc,Rd ...	mov Rb,Rd add Rb,Re,Re	mov Ra,Rb nop
6	add Rc,Rd ...	add Rc,Rb nop
7		mov Rb,Re nop
8		add Rd,Re nop
9		add Rc,Rd ...

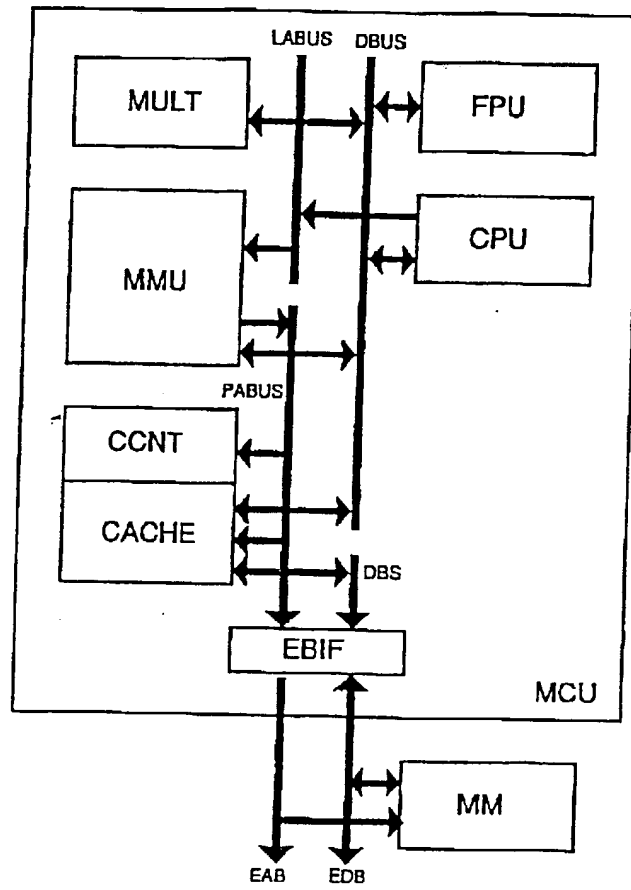
【図15】

図15



【図 14】

図 14



Real Time Debug for System-on-Chip Devices

Authors:

Richard York, ASIC Designer
John Sharp, Development Systems Group
ARM Ltd., Cambridge, UK
June 1999

Debugging software running in a highly integrated system-on-chip (SOC) device at frequencies in excess of a 100MHz poses some significant engineering challenges. Traditional software debug tools such as in-circuit emulators (ICE) or logic analyzers have relied on having access to most of the signals of a microprocessor. This is not the case when the microprocessor core is deeply embedded in an SOC. In an extreme case, there may be no core signals visible on the pins of the chip. The lack of signals is not the only problem that traditional methods have difficulty overcoming. As frequencies exceed 100MHz, any additions to data path lengths can cause the skew of signals to such an extent that an incorrect representation of processor activity occurs.

The pin-out problem can be overcome by using bondout versions of the SOC, which provide all the signals needed. Bondouts take on one of two forms: either as an exact replica of the final SOC or as an implementation of a common subset of the functionality for a product range. Both have their problems. An exact replica is likely to be of use for only one product, therefore new bondouts are needed for each new project. A subset requires further logic to be added around the bondout to provide the functionality of the SOC, so it will most likely behave differently to the final chip. The use of bondout technology always adds delay and cost to the design cycle. The additional work required is technically challenging (difficulty of routing signals off chip at maximum frequency) thus diverting technical expertise away from the main objective.

There is a strong market need for the same functionality provided by traditional methods to be available for SOC solutions. The features provided by an ICE and supporting software can be broadly divided into five major categories:

- 1) Breakpointing and stepping code
- 2) An historical non-intrusive trace of instruction flow and data accesses
- 3) Non-intrusive access to registers and memory
- 4) Emulation memory
- 5) Code coverage and profiling

Breakpointing and stepping application code allows users to run the application code to a given point in code and then stop the processor. At this point the user has the option of examining or changing memory or register contents, stepping or restarting the application.

The really difficult bugs to track down are those that occur in situations where there is an unforeseen and hence unpredictable interaction between the application software and hardware. These bugs can be intermittent and usually only occur when the system is running at full speed; simply starting, stopping or stepping the processor does not expose the problem. An historical non-intrusive trace of instruction flow and data accesses can provide the extra information needed to identify the bug. For example, an application crashes during an interrupt routine. The result of the crash is that a memory protection fault occurs; the cause can not be found using breakpoints and single stepping methods. The user sets up the trace filter facility to collect trace data only during the interrupt routine, and the trigger to stop tracing when the protection fault occurs. The filter facility limits the amount of information that has been traced and analyzed. The trigger ensures that the trace information around the bug has been captured and not over written. As trace buffer depths are finite, these features are important to ensure the buffer is only filled

with relevant information. They also save time by limiting the information that needs to be analyzed to find the bug. Trigger and filter conditions can be changed to refine what trace data is captured and when.

Another key feature is the ability to change a processor's registers or memory without stopping the system. This is very useful in electro-mechanical applications. For example, when debugging an engine management system, the ability to change the control parameters of the engine without having to stop it will save time.

Many ICEs provide emulation memory that can be forcibly switched into a processor memory map. This is achieved by monitoring all processor accesses; when an access is made to a region that has been designated to be emulation memory, the access is re-routed. One of the major uses for this memory is the replacement of ROM during development, enabling the user to modify code that would normally be resident in ROM, allowing for quicker code development cycles.

The most recent addition to the debug toolset is the use of code coverage and analysis tools, which provide users with several useful benefits:

- Prove test coverage
- Reduce code size
- Improve code performance
- Provide minimum and maximum execution times for an algorithm

The code coverage and analysis tools are usually resident on the host controlling the ICE or logic analyzer; the information required is provided by the trace facility. These tools set the trace trigger and filter functions, and then use the captured data to provide the user with the relevant information.

All the above functions are features of current ICEs, which are now required in the SOC world if these systems are to be efficiently debugged. The ARM solution puts the real time components of in-circuit emulation into the SOC. The advantages of this approach are:

- Debug solution running at full processor speed
- Debug of final product
- Scalable solution for multiprocessor devices
- Standard tools for all ARM core-based SOC's
- Easy, reliable and small interconnect to target
- Low cost

We use three solutions to provide this functionality:

- EmbeddedICE logic
- Real Time Trace
- Real Time Monitor

The EmbeddedICE logic, which is an integral part of an ARM, contains breakpoint registers that compare the value on the core address, data and control busses against values programmed into the registers. For example, the logic may be programmed to generate a breakpoint when an instruction is loaded from a particular address or a particular data value is stored to a given location. When a breakpoint occurs the processor will be stopped and will then enter debug state. Once the core is in this state, memory and register contents can be examined or modified, images can be loaded, code can be stepped or execution restarted. The EmbeddedICE logic provides all of the standard run control debug features.

The Real Time Trace solution for ARM cores embedded within an SOC is made up of:

- The Embedded Trace Macrocell

- The Trace Port Analyzer
- The Trace Debug Tools

These three elements provide the capability to trace instructions and data accesses in real time. The Trace Macrocell monitors the ARM core busses and passes compressed information via the trace port to the Trace Port Analyzer. The Analyzer is an external device, which stores the information from the trace port. The trace information is compressed so that the analyzer does not need to capture data at the same bandwidth as an analyzer monitoring the core busses directly. This has the benefit of either lowering the cost or increasing the amount of processor activity that can be traced. The debug tools retrieve the data from the analyzer and reconstruct an historical view of the processor's activity, with data accesses to memory interleaved. The trace display includes full symbol information and links to the source code being debugged, allowing rapid understanding of the trace data. The trigger and filter logic, which is a part of the Embedded Trace Macrocell, is configured via the JTAG port.

The Real Time Monitor provides two major functions with minimal intrusion on the application execution time:

- 1) The debug of foreground tasks while interrupts continue
- 2) The ability to read and write memory without stopping the processor

With the EmbeddedICE logic described above, a breakpoint forced the processor into debug state and stopped the processor clock. However, with the recently announced ARM9E and ARM10 cores, enhanced EmbeddedICE-RT logic allows a breakpoint to either stop the processor or generate an exception. If an exception is generated, the processor is vectored to a small monitor program (less than 2KB code and data) resident in target memory that provides full debug functionality. Interrupts are still enabled, allowing application interrupt handlers to

run and commands from the debugger to be received via JTAG. These commands provide full debugging of the foreground task: it can be stepped or restarted, further breakpoints can be set, and memory read and written.

To read or write memory without stopping the processor, a similar mechanism is used. The user selects a memory location to be read or written. The debugger sends commands via JTAG to the monitor program, which momentarily interrupts the application. The monitor program carries out the required operation; control is then returned to the application.

In SOC's the common use of Flash memory and the execution of ROM code from faster on-chip RAM alleviate the need for emulation memory. Fast code download to the memory is necessary to reduce design cycles. Using JTAG and the EmbeddedICE logic, ARM's Multi-ICE product achieves download speeds of 120KB/s, hence a 1MB application can be downloaded to on-chip RAM in eight seconds. Multi-ICE can also reprogram on-chip Flash memory.

With the availability of analyzer buffer depths of up to 40 million instructions, ample trace information is provided by ARM's Real Time Trace solution for use by code coverage and analysis tools.

ARM's debug solution provides all the functionality of a traditional ICE for SOC's, with no external visibility of core signals running at frequencies well in excess of 100MHz. This solution is applicable to all ARM core-based designs available from any of the ARM semiconductor partners who incorporate the Embedded Trace Macrocell in their SOC devices.

###